BTeV Trigger/DAQ Innovations

Margaret Votava on behalf of the BTeV Trigger and Trigger Groups

***Abstract*—The BTeV experiment was a collider based high energy physics (HEP) B-physics experiment proposed at Fermilab. It included a large-scale, high speed trigger/data acquisition (DAQ) system, reading data off the detector at 500 Gbytes/sec and writing to mass storage at 200 Mbytes/sec. The online design was considered to be highly credible in terms of technical feasibility, schedule and cost. This paper will give an overview of the overall trigger/DAQ architecture, highlight some of the challenges, and describe the BTeV approach to solving some of the technical challenges.**

At the time of termination in early 2005, the experiment had just passed its baseline review. Although not fully implemented, many of the architecture choices, design, and prototype work for the online system (both trigger and DAQ) were well on their way to completion. Other large, high-speed online systems may have interest in the some of the design choices and directions of BTeV, including (a) a commodity-based tracking trigger running asynchronously at full rate, (b) the hierarchical control and fault tolerance in a large real time environment, (c) a partitioning model that supports offline processing on the online farms during idle periods with plans for dynamic load balancing, and (d) an independent parallel highway architecture.

***Index Terms*—Data acquisition, large-scale systems, real time systems, triggering.**

I. Introduction

THE proposed BTeV detector [1] consisted of 6 separate subdetectors: pixel, silicon strips, straw tubes, RICH, EMCAL and muon with the pixel detector dominating the channel count (see Table 1).

Table 1. Channel Count

Subsystem	Channels	DCB Subsystems
Pixel	23M	10
Strips	118K	2
Straws	54K	6
RICH	144K	4
EMCAL	10K	2
Muon	37K	4

Manuscript received June 5, 2005. This work was supported in part Operated by Universities Research Association Inc. under Contract No. DE-AC02-76CH03000 with the United States Department of Energy.

M. Votava is with the Fermi National Accelerator Laboratory, Batavia, IL 60510 USA (phone: 630-840-2625; fax: 630-840-; e-mail: Votava@fnal.gov).

The overall detector was designed to run with a 396nsec (2.5 MHz) beam collision (a.k.a crossing or event) rate. The timing system was designed to deliver a 7.5 MHz clock to allow for calibration data to be taken between crossings. See Table 2 for the data rates at each trigger level.

Table 2. Event Rates

	Frequency	Event Size	Data Rate
Into L1	2.5 MHz	200 KB	500 GB/sec
Into L2/L3	50 KHz	250 KB	12.5 GB/sec
Into archival	2.5 KHz	80 KB	200 MB/sec

Fig. 1 shows the overall architecture of the BTeV trigger and DAQ.

There are several points in Fig. 1 worth noting. First, the detector was unique in that all the data was brought off the detector and digitized in subdetector-specific front end boards. This data was sent via point to point copper cable to data combiner boards (DCBs) before being sent to the first level trigger over optical links. The DCB design was common across subdetectors[1]. This architecture benefited the design in that 1) much of the L1 trigger could be done in software which allowed for many commodity components 2) the DCBs provided a single entry point into the trigger/DAQ that could be centrally designed and maintained, reducing the long term support load.

Secondly, data collected in the DCBs were routed to 8 independent, parallel paths called *highways*. This design reduced the control overhead on each particular highway and grouped data coming out of the L1 buffers into larger packets for better network performance.

Thirdly, the Level 2 and 3 trigger (L2 and L3, respectively) decisions were made on the same L2/L3 farm. Lastly, the baseline of the BTeV architecture defined logging data to large disk farms using dCache[2] as the underlying data storage support software.

The following sections will spell out some of the features of the online system: highway architecture including (II), fault tolerance (III), and partitioning (IV).

[1] More precisely, the output to the high speed optical links was in common. There were two variants to the input side. One for FPIX front end boards (Pixel and Strip detectors) and one for nonFPIX boards. See page **Error! Bookmark not defined.** for more detail.

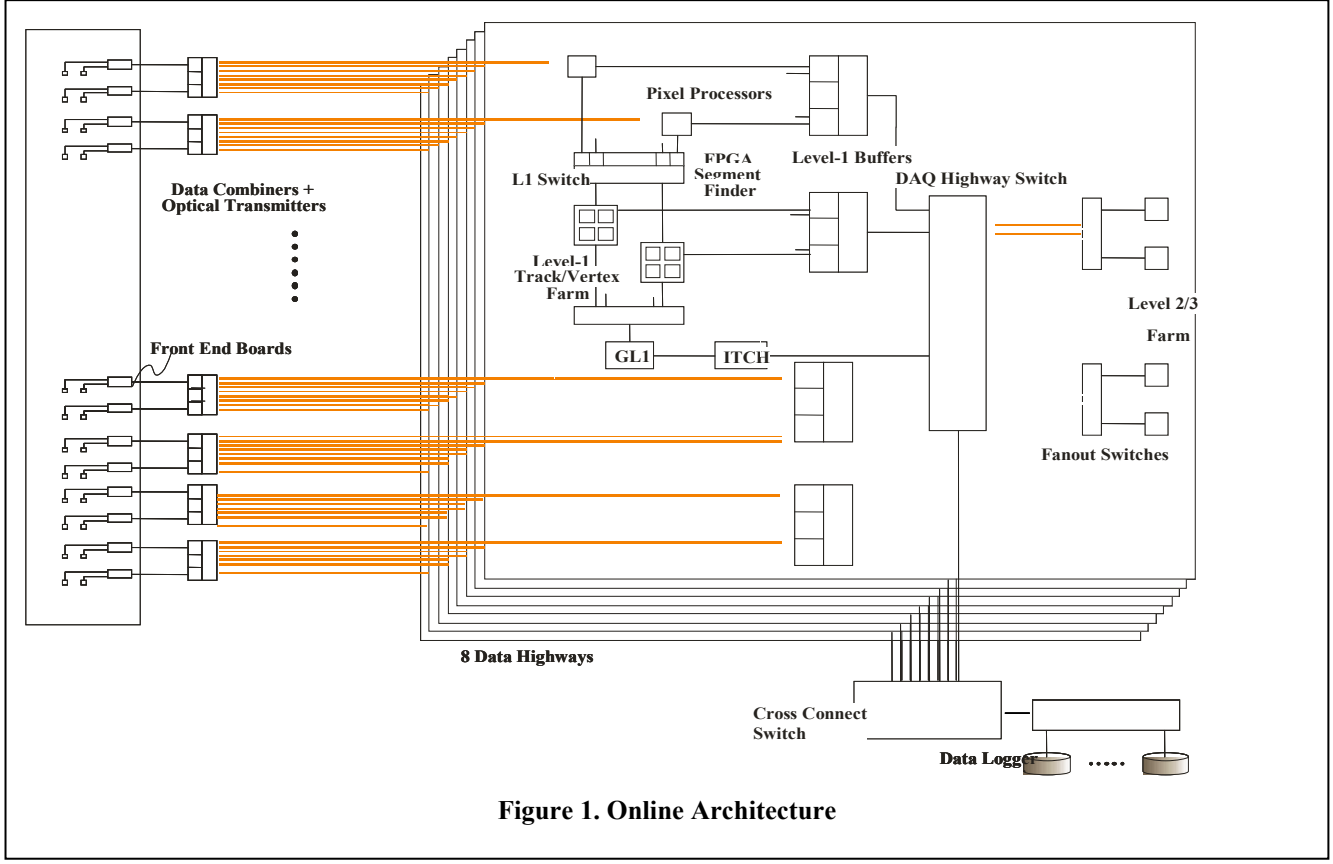

Figure 1. Online Architecture

II. HIGHWAY ARCHITECTURE

A. DCB

The DCBs were a custom component and interface between the subdetectors' front end electronics and the common data readout system. Two flavors of this board existed – one for the FPIX readout (Pixel and Strip detectors) and one for everyone else. For the purposes of this paper, they can be considered the same with only a variation on the configuration of the input ports and rates.

DCBs were the place in the online architecture that split the detector data into highways – eight parallel and independent data streams each processing 1/8th of the detector data. The original DCB design routed a crossing at a time. This resulted in a complex routing table algorithm to factor out any periodicity in the tevatron. The DCB sent data for a given crossing sent to one and only one highway.

Fig. 2 shows the data and control flow in/out of a DCB. Each board contained a commercial CPU and fast Ethernet interface for control and low speed data readout for diagnostics and commissioning. The DCBs received precise clock information from the timing system We imagined being able to reset/reconfigure "live" on a future crossing. E.g, if a catastrophic highway failure occured, the DCBs could be sent a control command to route to 7 highways instead of 8 starting at crossing number "N".

We were investigating the possibility of routing data a turn at a time. This alternative approach would simplify the routing, but increase the length of time data would live in the DCB, exposing it to a higher single event upset rate. It would also mean that data from FPIX sources would need to be time ordered, a functionality that was being designed into the pixel L1 trigger. Turn routing also had the advantage of being self balancing. That is, the traffic carried over a single highway was the same (on average) for each of the highways. This was true no matter how many highways were in service. If a highway dropped out, the remaining highways would have absorbed the load evenly.

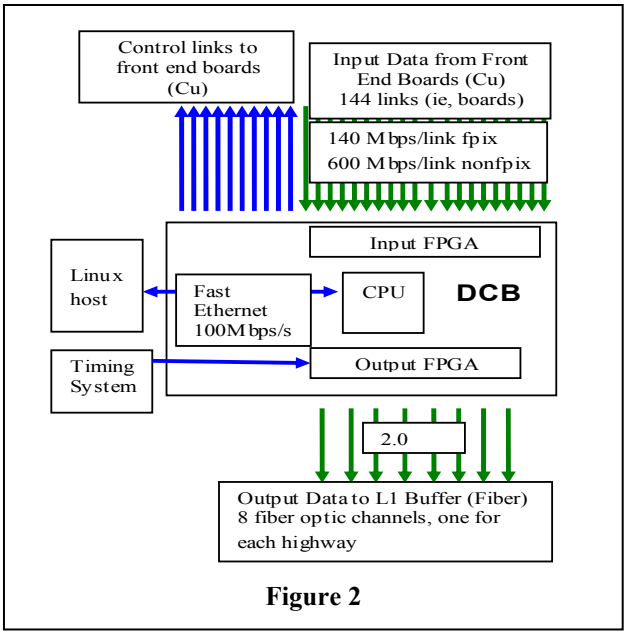

Figure 2

B. L1 Trigger

One particular highlight of the BTeV design is a commodity based L1 tracking trigger running asynchronously at full rate. As it is described in detail in a

paper[3] by M. Wang, it will only be summarized here.

Beam crossing data from low-level FPGA segment trackers were routed through a switching fabric to L1 worker nodes. L1 worker nodes were commodity processors similar to L2/L3. Each of the highways contained one L1 Infiniband switch. The segment tracker nodes (about fifteen) served as input to the switch. The data at the inputs are distributed to a small farm (about thirty-three) of L1 worker nodes. Data throughput simulations indicated that the output from each of the segment trackers, including excess capacity, totaled about 167 MB/s. The estimated total capacity needed per highway on the input side of the switch was 2.5 GB/s. Commercial off-the-shelf Infiniband switches were capable of handling this load. Due to fixed message latencies (between 4 to 8 us) in this type of switching fabric, small messages led to undesirable performance. Almost all the work of transferring data through the switch was done by the HCA, leaving the CPU free to processing the crossing data. The Infiniband switching fabric had enough capacity to handle the additional load necessary to send L1 worker-node results to Global Level-1 (GL1).

C. *Level 1 Buffers*

Data from the DCBs were routed over point to point optical links into L1 buffers, another custom electronics component. Each L1 Buffer could communicate with 24 DCBs (two crates worth). It's this unit that we will talk about in the partitioning section regarding resource reservation.

The primary responsibility of an L1 buffer was to buffer the detector data for a long enough time to make an L1 trigger decision and keep accepted events available until transferred to the L2 trigger. There were 24 L1 buffers per highway with 3G of memory each for an aggregate total of ~0.5 TBytes of memory, or roughly 1 second of data.

Data which were needed for trigger decisions (from pixel and muon detectors) were fed into the segment preprocessors in the L1 trigger. The L1 worker nodes also function as L1 Buffers serving L2 the results from L1 trigger algorithms.

Data entered the L1 buffer though an input FPGA and were stored in a circular buffer. After an L1 accept was received, all data for that subevent were copied into a smaller 1GB memory area that was used to transfer accepted events to the downstream L2 trigger farm. No special information was needed from the ITCH (Information Transfer Control Hardware) regarding L1 rejects; the data was simply overwritten in the circular buffer.

D. *L2/L3 Trigger*

From the L1 buffers, data were sent over a GBit Ethernet link to the to the L2/L3 trigger farm. The L2/L3 trigger software was running on the same hardware with the distinction being which physics algorithm would have been currently active. L2 was not required to make the event persist – it moved directly through memory to the L3 algorithms. L3 was responsible for the final event building.

The farm consisted of commodity processors and was also split into highways. Highways were self contained with lower bandwidth channels connecting them. Each highway consisted of 96 worker nodes, each with dual CPUs.

To reduce the control overhead and complexity of the software, we designed the event building switch with enough capacity to send a complete event to each L2 node. Each CPU box was referred to as a worker node. Worker nodes would declare themselves to a particular partition, ie, trigger list (see section on partitioning) and notify the ITCH when they were ready for data. The ITCH would assign them a particular crossing number. All data from that crossing would be sent to that worker node.

Worker nodes themselves were grouped into manageable units in a highway. Each group was controlled by a regional manager consisting of 12 worker nodes. A regional manager was responsible for configuring its associated worker nodes, fanning out control commands and collecting status, caching DBMS data (eg, various versions of the trigger algorithms and calibration data), and handling regional faults.

III. FAULT TOLERANCE

The BTeV trigger performed sophisticated computations using large ensembles of FPGAs and conventional microprocessors. This system would have had on the order of 5000 computing elements and many network switching elements. The need for fault-tolerant and fault-adaptive software, as well as flexible computing techniques and software to manage this huge computing platform had been identified as one of the more challenging aspects of this project.

As a response to this challenge, the Real Time Embedded Systems (RTES) project group was formed and funded through a 5 year NSF grant. This research group is a collaborative effort between electrical engineers, computer scientists and high energy physicists. The group is researching ways to increase the reliability of high-performance, heterogeneous, real-time systems.

The BTeV trigger was used as a model system for RTES. One reason for this is because of the overall asynchronous nature of this complex system, where no "hard failure" occurs if a specific computation is not delivered in time; only a temporary degradation of the performance is observed. Example observed failures are

computations takes a bit longer than expected, or, bunch crossings declared lost, or "rejected" if the computation takes too long. Our understanding of the nature of interactions indicated that timeouts are expected to occur expected to occur.

The RTES project has embraced a multi-aspect approach to large, real-time system modeling and fault specification and adaptation. Model integrated computing methods are used within a graphical modeling environment to describe system components and their physical relationships, messages exchanged by software processes, run control state machines, user interface definitions, and custom ARMOR elements. Metamodels define the domain-specific graphical languages for each of these. ARMORs (adaptive reconfigurable mobile objects for reliability) provide intrinsically reliable high level process oversight, with the ability to stop/restart applications in-place (on the same node), or to relocate the work to an available alternative resource. VLAs (very lightweight agents) provide low impact, low level detection of fault conditions.

These techniques have been demonstrated in a prototype of the BTeV L1 embedded processor farm, as well as in a prototype of the L2/3 commodity computer farm [3].

IV. Partitioning

Partitioning of the detector was defined to be running multiple independent data acquisition systems in parallel. This is not to be confused with the highway concept which was the physical implementation of parallel data streams. A partition is a logical concept and would have spanned across highways.

The value of partitioning, and when to partition the detector are different depending on the phase of the project i.e., partitioning needs during commissioning (testing the subdetectors in parallel) may be different than when testing new L3 trigger algorithms while taking physics quality data. Partitioning allowed spare cycles on the online trigger farm(s) to be used for other offline processing when beam is off or luminosity low without drastic change to the system configuration. The BTeV L2/L3 farm contained significant processing powe, and partitioning could provide a means to increase the utilization of this resource.

Partitioning was strictly a logical concept which needed to be mapped onto the physical implementation of the online system. The online DAQ/trigger was to be constructed in 2 stages with roughly 50% of capacity at each stage. The first stage consisted of 4 highways. The second stage was installed the next fiscal year. Even within a stage, individual highways were commissioned one at a time. The logical concept of partitioning needed to support running multiple partitions on a single highway (when only 1 was constructed) as well as the final system with 8 highways

The parallel highway architecture and dynamic reloading of DCB routing tables allowed for much flexibility in configuring partitions and rules were being established to limit to scope, function, and definition of partitioning. We had not reached a consensus on many details regarding partitioning at the time of termination. What we will discuss here are some of the ideas and the directions in which we were headed.

First we imagined that running a partition involved the following steps: 1) selecting/reserving [a subset of] electronics to be read out 2) Defining how much L2/L3 trigger processing power was needed and reserving those resources, 3) initializing the hardware, 4) collecting the data, and 5) freeing the resources

One of our original ideas for a possible use case was for a physicist to select the strips and straw front end crates, request 50 Mflops of L2/L3 nodes for processing, and then let software map this out onto a physical implementation. Depending on how many nodes might be needed, the layout may be to run on a single highway or to route to *n* highways. We rejected this idea because it could have been confusing to the end user to understand exactly where data is flowing. We ultimate developed a proposal that was a balance between flexibility and ease of understanding for the end user.

This approach imposed the following constraints: a) the hardware for the L1 trigger on a particular highway could not be partitioned, but could hold trigger tables for multiple partitions b) the smallest source unit that could be reserved was a single L1 buffer which corresponded to as many as 24 DCBs c) a worker node could belong to one and only one partition d) L2 worker nodes connected to the same regional manager could not span [online] partitions.

Because of the sheer number of electronics involved in the Trigger/DAQ, we were developing the idea of the L1 trigger and active highways always being available as a shared resource. A human run coordinator would establish the overall online configuration for a period of time (day/week/month) and coordinate the data taking runs during this period. This person would have the understanding of which configurations could support multiple overlapping runs. This stable configuration period had a fixed and predefined set of allowable highways. Subdetector groups still had to select the specific electronics to read out. These front end electronics could be reserved for read/write or readonly (ie, can't be reset or initialized). It was the responsibility of the run coordinator to schedule the detector so that users could get write access as needed. Partitions could come and go then, by adding/removing trigger tables as necessary.

An example would be the run coordinator making 4 highways available for the next two days. The pixel group could reserve the pixel front end electronics and associated L1 buffers for read/write, and load the pixel trigger table. Crossings would be distributed to all 4 highways. Online software would assign specific L2/L3 nodes to this partition

as constrained by the run coordinator. The silicon strip group could come and reserve silicon electronics for read/write and pixel for read only and load a second set of trigger tables. Again, the software would assign L2/L3 worker nodes specifically to this partition. If a given crossing passed the L1 trigger for both partitions, it could be routed to worker nodes in both partitions or split between the two partitions in a predefined scalar. This was also still being discussed in the collaboration.

Partitioning became an obvious solution when discussing the problem of how to utilize spare online cycles for offline. The pure Computer Scientists involved in RTES promoted real time scheduling on the worker nodes to maximize CPU utilization, but they were outweighed by the opinion that a particular worker node should be single tasking for an operation the was more manageable and easier to understand..

Nodes could manually be moved between online and offline partitions, but would be automatically shifted to offline as luminosity in the tevatron dropped. The automatic decision to migrate nodes between partitions would be influenced by metrics measured by RTES as well as an overall luminosity profile that would be loaded at the start of a run.

V. Conclusion

Thanks to the efforts of many talented people in the collaboration and at the lab as well as extremely helpful comments from the many reviews, the DAQ and trigger groups in BTeV were able to develop a highly credible online architecture. This architecture was believable in terms of cost and schedule with well understood, and mimimal risks.

Key elements in the success of the design were developing an architecture which included a) the split into independent highways which reduced individual bandwidth requirements, b) maximized the number of commodity components (switching fabric, trigger farms) which lowered cost (especially labor) and risk and well as allowed for a more plug and play upgrade path, c) minimized the variability between custom boards, *i.e.,*, the DCBs were common across all detectors, and d) capitalized on using large amounts of computing resources during periods of accelerator downtime.

References

[1] BTeV Collaboration, BTeV Proposal and TDR. Available: http://www-btev.fnal.gov/public/hep/general/proposal/index.shtml

[2] dCache

[3] M. Wang, " A Commodity Solution Based High Data Rate Asynchronous Trigger System for Hadron Collider Experiments", Proceedings of the the IEEE Real Time Conference 2005, Stockholm, Sweden, June 2005, to be published.

[4] M. J. Haney, et al., "The RTES Project - BTeV, and Beyond", Proceedings of the the IEEE Real Time Conference 2005, Stockholm, Sweden, June 2005, to be published.